**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**
**BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

| | |
|---|---|
| In re Application of: Boris PECHENY | |
| Application No.: 09/263,068 | Group Art Unit: 2172 |
| Filed: March 8, 1999 | Examiner: Fleurantin, J. |
| Attorney Docket: 50277-0164<br>Client Docket: OID-1997-039-01 | |

For: LEXICAL CACHE

## APPEAL BRIEF

Honorable Commissioner for Patents
Washington, D.C. 20231

Dear Sir:

This Appeal Brief is submitted, in triplicate, in support of the Notice of Appeal filed January 22, 2004.

## I. REAL PARTY IN INTEREST

Oracle International Corp. is the real party in interest.

## II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals and interferences.

## III. STATUS OF THE CLAIMS

Claims 1-5, 16-20, 31-35, and 37 are pending in this appeal. Claims 6-15 and 21-30 have been allowed. This appeal is therefore taken from the final rejection of claims 1-5, 16-20, 31-35, and 37 on October 21, 2003.

## IV. STATUS OF AMENDMENTS

No amendment has been filed after the final rejection.

## V. SUMMARY OF THE INVENTION

The present invention addresses problems associated with using conventional relational database index structures for applications performing text analysis. Relational databases store information in collections of tables, in which each table is organized into rows and columns, and index structures provide an easily searched mapping between row identifiers and key values derived from a column of the corresponding row. (Spec., p. 1:7-15)

For applications performing text analysis, the majority of accesses are highly skewed to relatively few rows of a database table (p. 1:17-18). Use of conventional relational database index structures to index this table, however, results in a sub-optimal performance for natural language processing applications, because the search time for very frequently accessed keys is about the same as the access time for rarely accessed keys (pp. 1:24-2:1) .

Accordingly, the present invention addresses the needs for a caching methodology such that searching a table of lexical data for frequently accessed keys results in search times that are significantly smaller than search times for rarely accessed keys. (p. 2:1-4) A lexical cache is provided comprising a collection of lexical containers, organized according to the length of the words (p. 3:2-3). This organization is advantageous because, when the lexicon is divided into

several subsets containing words of the same length, the subsets with the shorter length words tend to have a greater the frequency of usage, relative to the number of words in the subset, than subsets with longer length words (p. 3:4-9).

One aspect of the invention is a computer-implemented method and a computer-readable medium bearing instructions for searching for a string in a lexical cache (p. 3:10-11). A key is generated based on the string, for example, by compression (FIG. 3, step 300). A lexical container, such as a hash table (FIG. 2, 220), is identified from a plurality of lexical containers (221) based on the length of the key (302), and the identified lexical container is searched for an entry associated with the string (304, p. 3:12-15). In one embodiment, each lexical container contains a maximum number of entries based on a function that includes a term that is inversely proportional to logarithm of a key length associated with the lexical containers, which advantageously allows for a lexical cache to be fine-tune based on characteristics of a natural language, which obeys Zipf's law. (p. 13:5-12)

## VI. ISSUES

Whether claims 1-5, 16-20, 31-35, and 37 are obvious under 35 U.S.C. § 103 based on *Li* (US 5,774,588) in view of *Rangarajan et al.* (US 5,706,365)?

## VII. GROUPING OF CLAIMS

The claims should not be regarded as all standing together since the claims recite respective limitations that render each of the claims separately patentable. For the purposes of this appeal, the following groups are recognized:

A. Claims 1-5, 16-20, 31-33.

B. Claims 33-34.

C. Claim 35.

D. Claim 37.

## VIII. ARGUMENT

The initial burden of establishing a *prima facie* basis to deny patentability to a claimed invention under any statutory provision always rests upon the Examiner. *In re Mayne*, 41 USPQ2d 1451 (Fed .Cir. 1997); *In re Deuel*, 34 USPQ2d 1210 (Fed. Cir. 1995); *In re Bell*, 26 USPQ2d 1529 (Fed. Cir. 1993); *In re Oetiker*, 24 USPQ2d 1443 (Fed. Cir. 1992). In rejecting a claim under 35 U.S.C. § 103, the Examiner is required to provide a factual basis to support the obviousness conclusion. *In re Warner*, 154 USPQ 173 (CCPA 1967); *In re Lunsford*, 148 USPQ 721 (CCPA 1966); *In re Freed*, 165 USPQ 570 (CCPA 1970). The Examiner is required to show that all the claim limitations are taught or suggested by the references. *In re Royka*, 180 USPQ 580 (CCPA 1974); *In re Wilson*, 165 USPQ 494 (CCPA 1970).

### A. INDEPENDENT CLAIM 35 IS NOT RENDERED OBVIOUS BY *LI* AND *RANGARAJAN ET AL.* BECAUSE NEITHER REFERENCE DISCLOSES A "LOGARITHM."

Reversal of the rejection of independent 35 is respectfully requested, because neither *Li* nor *Rangarajan et al.* teach or otherwise suggest the limitations of the claim. For example claim 35 recites that the lexical containers are "each configured to contain a respective maximum number of entries based on a function that includes a term that is **inversely proportional to a logarithm** of a key length associated with the lexical containers." No disclosure of a function inversely proportional to logarithm can be found in the applied references.

In fact, the Examiner's reasoning in support of the rejection recognizes that "the combined teachings of Li and Rangarajan" faith teach the inversely proportional to a logarithm feature, because they need to be further modified to include that feature:

> It would have been obvious to a person of ordinary skill in the art at the time the invention was made to **modify the combined teachings of Li and Rangarajan** with allocating a plurality of lexical containers each configures [*sic*] to contain a respective maximum number of entries based on a function that includes a term that is inversely proportional to a logarithm of a key length associated with the lexical containers. (Final Office Action, p. 8, underlined emphasis original; bold emphasis added)

The Examiner expressly admitted, correctly, that "Li does not explicitly disclose, 'allocating a plurality of lexical containers to contain a respective maximum number of entries based on a function that includes a term that is inversely proportional to a logarithm of a key length associated with the lexical containers'." (p. 8, emphasis original) As for *Rangarajan et al.*, the Examiner's lengthy analysis is most notable for the fact that it does not address the inversely proportional to a logarithm limitation at all:

> However, Rangarajan discloses the page indexing module 127 then creates the word key for the word [*sic*] this includes determining 1011 the number k of n-grams for the word, the number k of n-grams for the word key is length [*sic*] of the word-2, (see col. 12, lines 38-48), the page key offset 511 is a [*sic*] offset to the start of the variable length page key 509 corresponding to the table entry, the page key size 513 is the total number of bytes in the corresponding page key 509 including all the entries for the n- gram and k values, (see col. 9, lines 30-34); further, in column 6, line 11-12, Rangarajan discloses the document list 225 stores a variable number of entries 311 up to [*sic*] maximum limit. (Final Office Action, p. 8)

This analysis leads to the maximum limit but stops short, right before it reaches how the maximum limit is configured. *Rangarajan et al.* at col. 6:10-14 states that the maximum limit is a constant $D_{max}$, which is 1,044,480 in the preferred embodiment, but fails to disclose that each lexical container is "configured to contain a respective maximum number of entries based on a

function that includes a term that is **inversely proportional to a logarithm** of a key length associated with the lexical containers" as recited in independent claim 35.

Therefore, neither *Li* nor *Rangrajan et al.*, the references of record, teach or otherwise suggest this feature, and the Examiner has not pointed to any coherent passage in the references that could disclose this feature, violating the mandate of the Administrative Procedures Act (APA) on the Patent Office to make the necessary findings and provide an administrative record showing the evidence on which the findings are based, accompanied by the reasoning in reaching its conclusions. See *In re Zurko*, 258 F.3d 1379, 1386, 59 USPQ2d 1693, 1697 (Fed. Cir. 2001); *In re Gartside*, 203 F.3d 1305, 1314, 53 USPQ2d 1769, 1774 (Fed. Cir. 2000).

### B. CLAIMS 1-5, 16-20, 31-34, AND 37 ARE NOT RENDERED OBVIOUS BY *LI* AND *RANGARAJAN ET AL.* BECAUSE NEITHER REFERENCE TEACHES A "SELECTING A LEXICAL CONTAINER ... BASED ON A LENGTH OF THE KEY."

The rejection of claims 1-5, 16-20, and 31-34, 37 should be reversed because neither *Li* nor *Rangajan et al.*, individually or in combination, teach the elements of the claims. For example, independent claims 1 and 31, as amended, recites (with added emphasis) as follows, with independent computer-readable medium claims 16 and 32 mirroring the steps of method claims 1 and 31, respectively:

> 1. A method of searching for a string in a lexical cache, comprising the computer-implemented steps of:
> generating a key based on the string;
> **selecting a lexical container from among a plurality of lexical containers based on a length of the key**, said lexical containers associated with respective key lengths and configured to hold respective maximum numbers of entries based on the respective key lengths; and
> **searching the selected lexical container** for an entry associated with the string based on the key,
> wherein at least one of the lexical containers is configured to hold a different maximum number of entries than at least another one of the lexical containers.

31. A method of storing a string in a lexical cache, comprising the computer-implemented steps of:

generating a key based on the string;

**selecting a lexical container from among a plurality of lexical containers based on a length of the key**, said lexical containers are associated with respective key lengths and configured to hold respective maximum numbers of entries based on the respective key lengths; and

**storing the string in an entry in the selected lexical container** based on the key,

wherein at least one of the lexical containers is configured to hold a different maximum number of entries than at least another one of the lexical containers.

This limitation is not found in either *Li* or *Rangarajan et al.* Rather, no data structure is found in *Li* or *Rangarajan et al.* that satisfy the claim limitations that a lexical container is selected "based on a length of a key." In particular, *Li* is directed to a method for comparing strings with entries of a lexicon using a fixed-length key. At step **120** of FIG. 1B, an incoming, unverified string **20** is processed to produce a signature vector **25** having a fixed length of 85 bits (col. 6:59-67). Then seven 12-bit partitions **30-36** of the signature vector **25** are hashed (FIG. 2, step **240**; col. 7:63–8:14) are hashed to generate a bucket address, with which entries of the *Li* lexicon are stored and retrieved. Associated with each bucket address is a linked list of buckets, in which each bucket holds a fixed number of entries, and the bucket list expands "to accommodate multiple signature vectors indexed to that address." (col. 7:56-58).

By contrast, *Li* does not disclose selecting a lexical container or even a hash table from among a plurality of such based on "a length of the key" as recited in the claims, because the key length is fixed to a constant length. The signature vector **25** key is fixed at 85 bits, and the partitions **30-36** of the signature vector **25** are a fixed 12 bits in length. Thus, there is no need nor motivation in *Li* to use that key's constant length to identify a particular lexical container or hash table.

Moreover, *Li*'s linked list of buckets at each bucket address, upon which the Examiner reads the recited "lexical container," fails to satisfy the limitations recited for the lexical container. For example, *Li* fails to disclose that the linked list of buckets at each bucket address is "associated with respective key lengths" or "configured to hold respective maximum numbers of entries based on the respective key lengths." The final Office Action, moreover, even acknowledged that "Li does not explicitly indicate a length of the key, and wherein at least one of the lexical containers is configured to hold a different maximum number of entries than at least another one of the lexical containers." (p. 3, emphasis original)

*Rangarajan et al.* is similarly deficient, describing a system and method for "indexing and retrieval of stored documents using a decomposition of words in the documents in n-grams, or linear word subunits." (Abstract). With specific reference to FIG. 10 for the indexing aspect of *Rangarajan et al.*, each word is expanded into *k* different n-grams (step **1011**), and an n-gram number is determined for each of the *k* n-grams (step **1013**). The group consisting of *k* and the *k* n-gram numbers is called a "word key" (col. 8:60–9:8). The n-gram number is used to index into a index page map within a bank index **223** (step **1023**) in order to set a corresponding bit in the index page map (step **1025**). After this loop is complete, the *Rangarajan et al.* system generates and writes a page key **509** (step **1031** or **1033**). The page key **509** denotes the "set of word keys for the all [*sic*] words on a page" (col. 9:7-8).

In retrieval, *Rangarajan et al.* discloses that a query word is decomposed into n-grams (FIG. 12, step **1205**), and, if an n-gram is present in a bank **217** (steps **1207**), the index page map for each page stored in the bank is checked for the n-gram (steps **1209** and **1211**). If the bit is set, a counter is set (step **1213**), and if the counter is sufficient to consider the page a hit, the page is flagged as a hit (step **1217**). As shown in FIG. 14, a page flagged as a hit is further searched, by

looping over each n-gram in the Query Word (step **1403**), visiting each word key in the page (step **1405**) to determine if a sufficient number of n-grams in the Query Word and the word key match (steps **1407-1413**).

By contrast, *Rangarajan et al.* fails to disclose "selecting a lexical container from among a plurality of lexical containers **based on a length of the key**." Rather, in *Rangarajan et al.* a page is flagged as a hit when n-grams of the Query Word are found in the page key **509**. Furthermore, *Rangarajan et al.* fails to suggest "said lexical containers associated with respective key lengths" because each page key **509** is associated instead with each page of a document that is indexed in the *Rangarajan et al.* system and has nothing to do with the length or number of n-grams in the Query Word. The page key size **513**, cited in the Office Action, does not support the rejection, because the page key size **513** is not "a length of a key" (rather it is the size of all the words keys in the page) nor the criterion used to select a page key **509** in the *Rangarajan et al.* system, which is matching n-grams.

The Examiner's rebuttal on p. 3 of the final Office Action is merely a word-for-word copy of the statement of the rejection as applied to all independent claims 35, regardless of their recitations.

### C. CLAIM 37 IS PATENTABLE OVER *LI* AND *RANGARAJAN ET AL.* BECAUSE NEITHER REFERENCE SHOWS A "SEARCHING ONLY THE SELECTED LEXICAL CONTAINER."

The rejection of dependent claim 37 is also improper because neither *Li* and *Rangarajan et al.* does all of its limitations. For example, claim 37 recites "searching only the selected lexical container," but all the pages that are flagged as hit in *Rangarajan et al.* are searched (see steps **1403** and **1405** of FIG. 14). The Examiner now contends that this feature is disclosed in *Li* at col. 6:42-46, but a quick glance to that passage shows nothing of the sort:

> Referring to FIG. **1A**, the lexicon is loaded into memory at step **100** of the flowchart. All lower case letters are mapped to their upper case letters, all between word spaces are stripped, and all non-alphanumeric characters are mapped to selected specific non-alphanumeric character (for example, "?").

In fact, the passage is not even remotely relevant to searching a selected lexical container, much less "searching only the selected lexical container," as recited in claim 37.

**D. CLAIM 33-34 ARE PATENTABLE BECAUSE NEITHER *LI* AND *RANGARAJAN ET AL.* DISCLOSE THAT "THE FIRST KEY LENGTH IS LESS THAN THE SECOND KEY LENGTH; AND THE FIRST LEXICAL CONTAINER IS CONFIGURED TO HOLD MORE ENTRIES THAN THE SECOND LEXICAL CONTAINER."**

Dependent claims 33-34 recite that "the first key length is less than the second key length; and the first lexical container is configured to hold more entries than the second lexical container." This reversal of this rejection too is respectfully requested.

Although the Examiner asserts that the second portion of the *Li* lexicon holds some of the entries of the first portion of the lexicon (pp. 5-6 citing col. 4:29-33 of *Li*), there is nothing in *Li* disclosing that the first portion of the lexicon is associated with a first key length that is **less than** the second key length for the second portion of the lexicon. In fact, the signature vector **25** key in *Li* is fixed at 85 bits.

## IX. CONCLUSION AND PRAYER FOR RELIEF

Appellants, therefore, request the Honorable Board to reverse each of the Examiner's rejections.

Respectfully Submitted,

DITTHAVONG & CARLSON, P.C.

3/22/2004
Date

Stephen C. Carlson
Attorney for Applicant(s)
Reg. No. 39929

10507 Braddock Rd, Suite A
Fairfax, VA 22032
Tel. 703-425-8516
Fax. 703-425-8518

## APPENDIX

1. (Previously Presented) A method of searching for a string in a lexical cache, comprising the computer-implemented steps of:

generating a key based on the string;

selecting a lexical container from among a plurality of lexical containers based on a length of the key, said lexical containers associated with respective key lengths and configured to hold respective maximum numbers of entries based on the respective key lengths; and

searching the selected lexical container for an entry associated with the string based on the key,

wherein at least one of the lexical containers is configured to hold a different maximum number of entries than at least another one of the lexical containers.

2. (Original) The method of claim 1, wherein the step of generating a key based on the string includes the step of compressing the string to produce the key.

3. (Original) The method of claim 2, wherein the step of compressing the string to produce the key includes the step of performing an n-gram compression on the string.

4. (Original) The method of claim 1, wherein the step of generating a key based on the string includes the step of using the string as the key.

5. (Previously Presented) The method of claim 1, wherein the step of selecting a lexical container includes the steps of:

generating a prefix based on the key; and

selecting the lexical container from among the plurality of the lexical containers based on the length of the key and the prefix.

6. (Previously Presented) A method of searching for a string in a lexical cache, comprising the computer-implemented steps of:

generating a key based on the string;

identifying a hash table from among a plurality of hash tables based on the length of the key, said hash table containing sequences of slots for holding entries associated with strings, each of said sequences of slots corresponding to a respective hash value, wherein at least one of the hash tables is configured to hold a different number of slots than at least another one of the hash tables;

computing a hash value based on the key; and

searching the hash table based on the hash value for a slot holding an entry associated with said string.

7. (Original) The method of claim 6, wherein the step of computing a hash value based on the key includes the step of computing the hash value based on the key and a prime number associated with the hash table.

8. (Original) The method of claim 7, wherein the step of searching the hash table based on the hash value includes the steps of:

indexing one or more fixed regions of the hash table, each of the fixed regions having the prime number of slots, based on the hash value to identify one or more respective slots; and

inspecting the one or more respective slots for a respective key value matching the key.

9. (Original) The method of claim 8, wherein the step of searching the hash table further includes the step of searching for the key in a linked list of slots stored in an expansion region of the hash table, if the key was not found in the one or more respective slots for the key.

10. (Original) The method of claim 6, further including the step of, if an entry for the string is not found at a first slot that corresponds to the hash value, but is found in a slot that belongs to a sequence of slots that correspond to keys that produce said hash value, then moving a relative position of the entry for the string within the sequence of slots toward the beginning of the sequence of slots.

11. (Original) The method of claim 6, further comprising the step of initializing a descriptor for the hash table, said descriptor storing a reference to the hash table and parameters for the hash table;

wherein the step of identifying a hash table includes the step of identifying a descriptor indicating the hash table and a prime number.

12. (Original) The method of claim 11, wherein the step of initializing a descriptor for the hash table includes the step of initializing a prime number for use in computing a hash value.

13. (Original) The method of claim 11, wherein the step of initializing a descriptor for the hash table includes the step of initializing a maximum number of slots for the hash table.

14. (Original) The method of claim 11, wherein the step of initializing a descriptor for the hash table includes the step of initializing a maximum length of the sequences of slots for the hash table.

15. (Previously Presented) A method of searching for a string in a lexical cache, comprising the computer-implemented steps of:

compressing the string to generate a key;

identifying a hash table from among a plurality of hash tables based on a length of the key, said hash table containing sequences of slots for holding respective key values, each of said sequences of slots corresponding to a respective hash value and a number of slots being based on a respective key length, wherein at least one of the hash tables is configured to hold a different number of slots than at least another one of the hash tables;

computing a hash value based on the key;

using said hash value to locate a beginning of the particular sequence of slots that correspond to said hash value;

searching the particular sequence of slots for a slot holding a key value matching the key; and

if a slot having a key value matching the key is found in the particular sequence of slots, but is not at the beginning of said particular sequence of slots, then moving a relative position of

the key value within the particular sequence of slots toward the beginning of the particular sequence of slots.

16. (Previously Presented) A computer-readable medium bearing instructions for searching for a string in a lexical cache, said instructions arranged, when executed by one or more processors, to cause the one or more processors to perform the steps of:

generating a key based on the string;

selecting a lexical container from among a plurality of lexical containers based on a length of the key, said lexical containers associated with respective key lengths and configured to hold respective maximum numbers of entries based on the respective key lengths; and

searching the selected lexical container for an entry associated with the string based on the key,

wherein at least one of the lexical containers is configured to hold a different maximum number of entries than at least another one of the lexical containers.

17. (Original) The computer-readable medium of claim 16, wherein the step of generating a key based on the string includes the step of compressing the string to produce the key.

18. (Original) The computer-readable medium of claim 17, wherein the step of compressing the string to produce the key includes the step of performing an n-gram compression on the string.

19. (Original) The computer-readable medium of claim 16, wherein the step of generating a key based on the string includes the step of using the string as the key.

20. (Previously Presented) The computer-readable medium of claim 16, wherein the step of selecting a lexical container includes the steps of:

generating a prefix based on the key; and

selecting the lexical container from among the plurality of the lexical containers based on the length of the key and the prefix.

21. (Previously Presented) A computer-readable medium bearing instructions for searching for a string in a lexical cache, said instructions arranged, when executed by one or more processors, to cause the one or more processors to perform the steps of:

generating a key based on the string;

identifying a hash table from among a plurality of hash tables based on the length of the key, said hash table containing sequences of slots for holding entries associated with strings, each of said sequences of slots corresponding to a respective hash value, wherein at least one of the hash tables is configured to hold a different number of slots than at least another one of the hash tables;

computing a hash value based on the key; and

searching the hash table based on the hash value for a slot holding an entry associated with said string.

22. (Original) The computer-readable medium of claim 21, wherein the step of computing a hash value based on the key includes the step of computing the hash value based on the key and a prime number associated with the hash table.

23. (Original) The computer-readable medium of claim 22, wherein the step of searching the hash table based on the hash value includes the steps of:

indexing one or more fixed regions of the hash table, each of the fixed regions having the prime number of slots, based on the hash value to identify one or more respective slots; and

inspecting the one or more respective slots for a respective key value matching the key.

24. (Original) The computer-readable medium of claim 23, wherein the step of searching the hash table further includes the step of searching for the key in a linked list of slots stored in an expansion region of the hash table, if the key was not found in the one or more respective slots for the key.

25. (Original) The computer-readable medium of claim 21, wherein said instructions are further arranged to cause the one or more processors to perform the step of, if an entry for the string is not found at a first slot that corresponds to the hash value, but is found in a slot that belongs to a sequence of slots that correspond to keys that produce said hash value, then moving a relative position of the entry for the string within the sequence of slots toward the beginning of the sequence of slots.

26. (Original) The computer-readable medium of claim 21, wherein said instructions are further arranged to cause the one or more processors to perform the step of initializing a descriptor for the hash table, said descriptor storing a reference to the hash table and parameters for the hash table;

wherein the step of identifying a hash table includes the step of identifying a descriptor indicating the hash table and a prime number.

27. (Original) The computer-readable medium of claim 26, wherein the step of initializing a descriptor for the hash table includes the step of initializing a prime number for use in computing a hash value.

28. (Original) The computer-readable medium of claim 26, wherein the step of initializing a descriptor for the hash table includes the step of initializing a maximum number of slots for the hash table.

29. (Original) The computer-readable medium of claim 26, wherein the step of initializing a descriptor for the hash table includes the step of initializing a maximum length of the sequences of slots for the hash table.

30. (Previously Presented) A computer-readable medium bearing instructions for searching for a string in a lexical cache, said instructions arranged, when executed by one or more processors, to cause the one or more processors to perform the steps of:

compressing the string to generate a key;

identifying a hash table from among a plurality of hash tables based on a length of the key, said hash table containing sequences of slots for holding respective key values, each of said sequences of slots corresponding to a respective hash value and a number of slots being based on a respective key length, wherein at least one of the hash tables is configured to hold a different number of slots than at least another one of the hash tables;

computing a hash value based on the key;

using said hash value to locate a beginning of the particular sequence of slots that correspond to said hash value;

searching the particular sequence of slots for a slot holding a key value matching the key; and

if a slot having a key value matching the key is found in the particular sequence of slots, but is not at the beginning of said particular sequence of slots, then moving a relative position of the key value within the particular sequence of slots toward the beginning of the particular sequence of slots.

31. (Previously Presented) A method of storing a string in a lexical cache, comprising the computer-implemented steps of:

generating a key based on the string;

selecting a lexical container from among a plurality of lexical containers based on a length of the key, said lexical containers are associated with respective key lengths and configured to hold respective maximum numbers of entries based on the respective key lengths; and

storing the string in an entry in the selected lexical container based on the key,

wherein at least one of the lexical containers is configured to hold a different maximum number of entries than at least another one of the lexical containers.

32. (Previously Presented) A computer-readable medium bearing instructions for storing a string in a lexical cache, said instructions arranged, when executed by one or more processors, to cause the one or more processors to perform the steps of:

generating a key based on the string;

selecting a lexical container from among a plurality of lexical containers based on a length of the key, wherein the lexical containers are associated with respective key lengths and configured to hold respective maximum numbers of entries based on the respective key lengths; and

storing the string in an entry in the selected lexical container based on the key, wherein at least one of the lexical containers is configured to hold a different maximum number of entries than at least another one of the lexical containers.

33. (Previously Presented) The method of claim 1, wherein:

a first lexical container of the lexical containers is associated with a first key length;

a second lexical container of the lexical containers is associated with a second key length;

the first key length is less than the second key length; and

the first lexical container is configured to hold more entries than the second lexical container.

34. (Previously Presented) The method of claim 31, wherein:

a first lexical container of the lexical containers is associated with a first key length;

a second lexical container of the lexical containers is associated with a second key length;

the first key length is less than the second key length; and

the first lexical container is configured to hold more entries than the second lexical container.

35. (Previously Presented) A method of providing a lexical cache, comprising the computer-implemented steps of:

allocating a plurality of lexical containers each configured to contain a respective maximum number of entries based on a function that includes a term that is inversely proportional to a logarithm of a key length associated with the lexical containers; and

searching for one of the entries associated with a string within one of the plurality of lexical containers corresponding to a key generated based on the string.

36. (Canceled)

37. (Previously Presented) The method of claim 1, wherein the step of searching the selected lexical container includes searching only the selected lexical container.